One Test

Jasen Jacobsen

The MITRE Corporation

Questions from ITSAC OVAL Workshop

Are there changes that would facilitate greater adoption?

- Need to understand the implementation challenges with OVAL today.
- Make OVAL easier to implement.

How do we scale more efficiently?

- Need to create component schemas for many more platforms.
- Schema development is community driven.
- No single team will be an expert in all platforms.

Can OVAL be made more easily extensible?

- The core schemas have been very stable.
- The component schemas change often.
- Component schema additions and revisions require OVAL Language revisions.
- Well documented with example extensions including defined criteria for a valid extension
  - Could the development of extensions the be federated?

What would make OVAL more maintainable?

- All tests are essentially identical.
- The meaning of <trustee_sid/> is defined 23 times in OVAL 5.8.
- At a minimum the meaning of a given object entity is defined three times.

Major Revision Possibilities

- Define only one test type

One Test Type

- 136 Tests Defined in Version 5.8.
- Tests associate an object with a state(s).
- Define how many items must satisfy the object and how many of those matching items must satisfy the state(s).

Test Example

```
<tests>
  <registry_test xmlns="…">
    <object object_ref="oval:org.example.oval:obj:1"/>
    <state state_ref="oval:org.example.oval:ste:1"/>
  </registry_test>
</tests>
<objects>
  <registry_object id="oval:org.example.oval:obj:1" version="1" xmlns="…">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
  </registry_object>
</objects>
<states>
  <registry_state id="oval:org.example.oval:ste:1" version="1" xmlns="…">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
    <type>reg_sz</type>
    <value>test</value>
  </registry_state>
</states>
```

Test Example

```
<tests>
  <registry_test xmlns="…">
    <object object_ref="oval:org.example.oval:obj:1"/>
    <state state_ref="oval:org.example.oval:ste:1"/>
  </registry_test>
</tests>
<objects>
  <registry_object id="oval:org.example.oval:obj:1" version="1" xmlns="…">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
  </registry_object>
</objects>
<states>
  <registry_state id="oval:org.example.oval:ste:1" version="1" xmlns="…">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
    <type>reg_sz</type>
    <value>test</value>
  </registry_state>
</states>
```

Schema Example

```
<xsd:element name="version_test" substitutionGroup="oval-def:test">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <oval:element_mapping>...</oval:element_mapping>
    </xsd:appinfo>
    <xsd:appinfo>
      <sch:pattern id="catos-def_version_test">
        <sch:rule context="catos-def:version_test/catos-def:object">
          <sch:assert test="...">the object child element must ...</sch:assert>
        </sch:rule>
        <sch:rule context="catos-def:version_test/catos-def:state">
          <sch:assert test="...">the state child element must ...</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="oval-def:TestType">
        <xsd:sequence>
          <xsd:element name="object" type="oval-def:ObjectRefType" />
          <xsd:element name="state" type="oval-def:StateRefType" minOccurs="0"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

Schematron Example

```
<xsd:element name="version_test" substitutionGroup="oval-def:test">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <oval:element_mapping>...</oval:element_mapping>
    </xsd:appinfo>
    <xsd:appinfo>
      <sch:pattern id="catos-def_version_test">
        <sch:rule context="catos-def:version_test/catos-def:object">
          <sch:assert test="@object_ref=ancestor::oval-def:oval_definitions/
            oval-def:objects/catos-def:version_object/@id">...</sch:assert>
        </sch:rule>
        <sch:rule context="catos-def:version_test/catos-def:state">
          <sch:assert test="@state_ref=ancestor::oval-def:oval_definitions/
            oval-def:states/catos-def:version_state/@id">...</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="oval-def:TestType">
        <xsd:sequence>
          <xsd:element name="object" type="oval-def:ObjectRefType" />
          <xsd:element name="state" type="oval-def:StateRefType" minOccurs="0"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
```

Schematron Example

```
<xsd:element name="version_test" substitutionGroup="oval-def:test">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <oval:element_mapping>...</oval:element_mapping>
    </xsd:appinfo>
    <xsd:appinfo>
      <sch:pattern id="catos-def_version_test">
        <sch:rule context="catos-def:version_test/catos-def:object">
          <sch:assert test="@object_ref=ancestor::oval-def:oval_definitions/
            oval-def:objects/catos-def:version_object/@id">...</sch:assert>
        </sch:rule>
        <sch:rule context="catos-def:version_test/catos-def:state">
          <sch:assert test="@state_ref=ancestor::oval-def:oval_definitions/
            oval-def:states/catos-def:version_state/@id">...</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="oval-def:TestType">
        <xsd:sequence>
          <xsd:element name="object" type="oval-def:ObjectRefType" />
          <xsd:element name="state" type="oval-def:StateRefType" minOccurs="0"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
```

What’s the Problem?

- Schema explosion
  - Many items with the same structure but different names.
  - Adds complexity
  - Potential for Error
  - Maintenance headache
  - Implementation headache

Proposal – One Test

- Change schema to provide a single test element that can be bound to many different objects and state(s)
- Tests become differentiated at the instance level (like definitions) rather than at the schema level

Benefits

- Simpler Schema
- Simpler Interpreter Implementation
- Less for content authors to remember

Challenges

- How to bind Objects to States?
- Loss of description in schema

```
<xsd:element name="regkeyauditedpermissions53_test" substitutionGroup="oval-def:test">
  <xsd:annotation>
    <xsd:documentation>The registry key audited permissions test is used to check the
audit permissions associated with Windows registry keys. Note that the trustee's audited
permissions are the audit permissions that the SACL grants to the trustee or to any
groups of which the trustee is a member. It extends the standard TestType as defined in
the oval-definitions-schema and one should refer to the TestType description for more
information. The required object element references a regkeyauditedpermissions53_object
and the optional state element specifies the metadata to check. The evaluation of the
test is guided by the check attribute that is inherited from the
TestType.</xsd:documentation>
```

Implementation

- Two options under consideration
  - Modify “test” element and TestType in oval-definitions-schema.xsd
  - Add “test” element to independent-definitions-schema.xsd
- Deprecate all existing tests

Option One

```
<xsd:element name="test" type="oval-def:TestType" abstract="true">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:complexType name="TestType">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <sch:pattern id="oval-def_test_type">...</sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="notes" type="oval-def:NotesType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="oval:TestIDPattern" use="required"/>
  <xsd:attribute name="version" type="xsd:nonNegativeInteger" use="required"/>
  <xsd:attribute name="check_existence" type="oval:ExistenceEnumeration"
    use="optional" default="at_least_one_exists"/>
  <xsd:attribute name="check" type="oval:CheckEnumeration" use="required"/>
  <xsd:attribute name="state_operator" type="oval:OperatorEnumeration"
    use="optional" default="AND"/>
  <xsd:attribute name="comment" type="oval:NonEmptyStringType" use="required"/>
  <xsd:attribute name="deprecated" type="xsd:boolean"
    use="optional" default="false"/>
</xsd:complexType>
```

Option One

```
<xsd:element name="test" type="oval-def:TestType" abstract="false">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:complexType name="TestType">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <sch:pattern id="oval-def_test_type">...</sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="object" type="oval-def:ObjectRefType" minOccurs="0"/>
    <xsd:element name="state" type="oval-def:StateRefType" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="notes" type="oval-def:NotesType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="oval:TestIDPattern" use="required"/>
  <xsd:attribute name="version" type="xsd:nonNegativeInteger" use="required"/>
  <xsd:attribute name="check_existence" type="oval:ExistenceEnumeration"
    use="optional" default="at_least_one_exists"/>
  <xsd:attribute name="check" type="oval:CheckEnumeration" use="required"/>
  <xsd:attribute name="state_operator" type="oval:OperatorEnumeration"
    use="optional" default="AND"/>
  <xsd:attribute name="comment" type="oval:NonEmptyStringType" use="required"/>
  <xsd:attribute name="deprecated" type="xsd:boolean"
    use="optional" default="false"/>
```

Option One Example - Good

```
<tests>
  <test id="oval:org.example.oval:tst:1" version="1" check="all"
    comment="An example."
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5">
    <object object_ref="oval:org.example.oval:obj:1"/>
    <state state_ref="oval:org.example.oval:ste:1"/>
  </test>
</tests>
<objects>
  <registry_object id="oval:org.example.oval:obj:1" version="1"
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
  </registry_object>
</objects>
<states>
  <registry_state id="oval:org.example.oval:ste:1" version="1"
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
    <type>reg_sz</type>
    <value>test</value>
  </registry_state>
</states>
```

Option One Example - Bad

```
<tests>
  <registry_test id="oval:org.example.oval:tst:1" version="1" check="all"
    comment="A [illegible]xample."
    xmlns="ht[illegible]org/XMLSchema/ova[illegible]#windows">
    <oval-def:object [illegible]l:obj:4"/>
    <oval-def:state sta[illegible]l:ste:3"/>
    <object o[illegible]rg.example.oval[illegible]
    <state sta[illegible]oval:org.example.oval:ste:1"/>
  </test>
</tests>
<objects>
  <registry_object id="oval:org.example.oval:obj:1" version="1"
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
  </registry_object>
</objects>
<states>
  <registry_state id="oval:org.example.oval:ste:1" version="1"
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
    <type>reg_sz</type>
    <value>test</value>
  </registry_state>
</states>
```

HS SEDI | MITRE
Homeland Security Systems Engineering and Development Institute

Option One

- Pros
  - Aligns with how definitions are declared
- Cons
  - XML authoring tools may give bad hints
  - Requires several Schematron rules to ensure correct Object and State elements are used

Option Two

- **Add “test” element to independent-definitions-schema.xsd**

Option Two

```
<xsd:element name="test" substitutionGroup="oval-def:test">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <sch:pattern id="ind-def_test">
        <sch:rule context="ind-def:test/ind-def:object">
          <sch:assert test="@object_ref=ancestor::oval-def:oval_definitions/
            oval-def:objects/*/@id">...</sch:assert>
        </sch:rule>
        <sch:rule context="ind-def:test/ind-def:state">
          <sch:assert test="@state_ref=ancestor::oval-def:oval_definitions/
            oval-def:states/*/@id">...</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="oval-def:TestType">
        <xsd:sequence>
          <xsd:element name="object" type="oval-def:ObjectRefType"/>
          <xsd:element name="state" type="oval-def:StateRefType" minOccurs="0"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

Option Two

- Pros
  - Familiar Implementation
  - Does not change core schema
- Cons
  - None?

Option Two Example

```
<tests>
  <test id="oval:org.example.oval:tst:1" version="1" check="all"
    comment="An example."
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#independent">
    <object object_ref="oval:org.example.oval:obj:1"/>
    <state state_ref="oval:org.example.oval:ste:1"/>
  </test>
</tests>
<objects>
  <registry_object id="oval:org.example.oval:obj:1" version="1"
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
  </registry_object>
</objects>
<states>
  <registry_state id="oval:org.example.oval:ste:1" version="1"
    xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Example</key>
    <name>Example</name>
    <type>reg_sz</type>
    <value>test</value>
  </registry_state>
</states>
```

Both Schema Options

- How to enforce Object and State pairings?
  - Schematron
  - Use element_mapping element

Element Mapping

```
<xsd:element name="registry_test" substitutionGroup="oval-def:test">
  <xsd:annotation>
    <xsd:documentation>...</xsd:documentation>
    <xsd:appinfo>
      <oval:element_mapping>
        <oval:test>registry_test</oval:test>
        <oval:object>registry_object</oval:object>
        <oval:state>registry_state</oval:state>
        <oval:item target_namespace="...#windows">registry_item</oval:item>
      </oval:element_mapping>
    </xsd:appinfo>
    <xsd:appinfo>
      <sch:pattern id="win-def_regtst">
        <sch:rule context="win-def:registry_test/win-def:object">
          <sch:assert test="@object_ref=ancestor::oval-def:oval_definitions/
            oval-def:objects/win-def:registry_object/@id">...</sch:assert>
        </sch:rule>
        <sch:rule context="win-def:registry_test/win-def:state">
          <sch:assert test="@state_ref=ancestor::oval-def:oval_definitions/
            oval-def:states/win-def:registry_state/@id">...</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
```

Element Mapping

- Schematron needs access to this element mapping lookup table.

- OPTION 1: Generate an element mapping document imported by the Schematron
  - This solution creates a dependency on an external file

```
<sch:let name="element_mapping_doc" value="document($em_doc_uri)"/>
```

- OPTION 2:Create assertion for each Object & State pair
  - Use XSLT to create Schematron from element mapping
  - This solution creates a lot of Schematron

Schematron Process – Option One

```
<sch:rule context="ind-def:test/ind-def:state">
...

<!-- Gather info about the referenced state -->
<sch:let name="state_ref" value="./@state_ref"/>
 <sch:let name="reffed_state" value="ancestor::oval-def:oval_definitions/oval-
def:states/*[@id=$state_ref]"/>
<sch:let name="state_name" value="name($reffed_state)"/>
<sch:let name="state_namespace" value="namespace-uri($reffed_state)"/>
...

<!-- Load the element mapping document. -->
<sch:let name="element_mapping_doc" value="document($em_doc_uri)"/>
...

<!-- Find the element_mapping for the object. There should be only one. -->
<sch:let name="element_map"
value="$element_mapping_doc//oval:element_mapping[string(oval:object) =
string($object_name)]"/>
...
<!-- NOTE: This does not take into account the namespaces of the object & state. But they should
be the same - ALWAYS. -->
<sch:assert test="$state_name = string($element_map/oval:state)">The referenced state should be
...</sch:assert>
```

Schematron Process – Option Two

```
<sch:rule context="ind-def:test/ind-def:state">
...

<!-- Gather info about the referenced state -->
<sch:let name="state_ref" value="./@state_ref"/>
<sch:let name="reffed_state" value="ancestor::oval-def:oval_definitions/oval-
def:states/*[@id=$state_ref]"/>
<sch:let name="state_name" value="name($reffed_state)"/>
<sch:let name="state_namespace" value="namespace-uri($reffed_state)"/>
...
<!-- The below can be broken down as:
     assert (this is not the 'A' I'm looking for) or ('B' has the value I'm expecting.)
     assert (this is not the state I'm checking) or (I found the object I'm expecting)
-->
<sch:assert test="not(($state_namespace='http://oval.mitre.org/XMLSchema/oval-definitions-5#windows'
   and ($state_name = 'registry_state'))
  or (($object_namespace='http://oval.mitre.org/XMLSchema/oval-definitions-5#windows')
      and ($object_name='registry_object'))">
...
</sch:assert>
```

Questions

- Is it a good idea to transition to the "one test"?
  - Does the current Test, Object, State approach still work?
    - Provides documentation
    - Encapsulates metadata
  - Does switching to one test create as many problems as it solves?
    - Loss of granularity
    - Adds complexity to Schematron generation

BACKUP SLIDES

What about unknown_test?

- Could be left as is
- Could be deprecated
  - Create an “UnknownCriterionType”
  - Create an “unknown_criterion” element.

HS SEDI | MITRE
Homeland Security Systems Engineering and Development Institute

Unknown Criterion

```
<xsd:complexType name="CriteriaType">
  ...
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element name="criteria" type="oval-def:CriteriaType"/>
            <xsd:element name="criterion" type="oval-def:CriterionType"/>
            <xsd:element name="unknown_criterion" type="oval-def:UnknownCriterionType"/>
            <xsd:element name="extend_definition" type="oval-def:ExtendDefinitionType"/>
        </xsd:choice>
...
</xsd:complexType>
<xsd:complexType name="UnknownCriterionType">
    <xsd:annotation>
      <xsd:documentation>The UnknownCriterionType complex type acts as a placeholder
        for tests whose implementation is unknown.</xsd:documentation>
      <xsd:documentation>The optional comment attribute provides a short description
        of any information that is known about the test</xsd:documentation>
    </xsd:annotation>
  <xsd:attribute name="comment" type="oval:NonEmptyStringType" use="optional"/>
</xsd:complexType>
```